Refine Search

Search Results -

Terms	Documents
L19 not @py>1999	12

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Wednesday, August 15, 2007 **Purge Queries** Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L20	L19 not @py>1999	12	L20
L19	L18 and file	491	L19
L18	L17 and pattern	568	L18
L17	L16 and match$	884	L17
L16	l15 and (text or words or text with string or text adj string or text near string)	1069	L16
L15	L14 and search	1275	L15
L14	("foreign assets control" or foreign and assets and control or foreign with assets with control)	2396	L14
L13	345/347	652	L13
L12	345.clas.	87378	L12
L11	707/104.1	8909	L11
L10	707/6	4511	L10

L9	707/3	11076	L9
L8	707.clas.	57568	L8
L7	705.clas.	52753	L7
L6	705/1	7479	L6
L5	705/43	793	L5
L4	705/42	813	L4
L3	705/39	2351	L3
L2	705/38	1284	L2
L1	705/35	3192	L1

END OF SEARCH HISTORY

First Hit Fwd Refs Previous Doc Next Doc Go to Doc#

Generate Collection Print

L20: Entry 6 of 12 File: USPT Dec 15, 1998

US-PAT-NO: 5850518
DOCUMENT-IDENTIFIER: US 5850518 A

TITLE: Access-method-independent exchange

DATE-ISSUED: December 15, 1998

INVENTOR-INFORMATION:

NAME	CITY	STATE	ZIP CODE	COUNTRY
Northrup; Charles J.	Old Bridge	NJ	08857	

APPL-NO: 08/353905 [PALM]
DATE FILED: December 12, 1994

INT-CL-ISSUED: [06] G06F 15/00

INT-CL-CURRENT:

TYPE	IPC	DATE
CIPS	H04 L 29/08	20060101
CIPS	H04 L 29/06	20060101

US-CL-ISSUED: 395/200.33; 364/242.94, 364/DIG.1
US-CL-CURRENT: 709/203

FIELD-OF-CLASSIFICATION-SEARCH: 395/800, 395/650, 395/200.03, 395/402, 395/680, 395/589, 395/590, 395/200.33, 364/242.94, 364/DIG.1
See application file for complete search history.

PRIOR-ART-DISCLOSED:

U.S. PATENT DOCUMENTS

Search Selected Search ALL Clear

	PAT-NO	ISSUE-DATE	PATENTEE-NAME	US-CL
☐	4835673	May 1989	Rushby et al.	364/200
☐	4835674	May 1989	Collins et al.	364/200
☐	4982344	January 1991	Jordan	364/521
☐	5021949	June 1991	Morten et al.	364/200
☐	5109515	April 1992	Laggis et al.	395/725
☐	5159592	October 1992	Perkins	370/85.7

☐	5185860	February 1993	Wu	395/200
☐	5239634	August 1993	Buch et al.	395/410
☐	5251205	October 1993	Callon et al.	370/60
☐	5265239	November 1993	Ardolino	395/500
☐	5278955	January 1994	Forte et al.	395/200
☐	5297249	March 1994	Bernstein et al.	395/156
☐	5301326	April 1994	Linnett et al.	395/700
☐	5307347	April 1994	Duault et al.	370/85.1
☐	5309437	May 1994	Perlman et al.	370/85.13
☐	5321542	June 1994	Freitas et al.	359/172
☐	5355476	October 1994	Fukumura	395/600
☐	5367619	November 1994	Dipaolo et al.	395/149
☐	5412784	May 1995	Rectschaffen et al.	395/200.03
☐	5475819	December 1995	Miller et al.	395/200.03
☐	5497463	March 1996	Stein et al.	395/200.03
☐	5510980	April 1996	Peters	364/406
☐	5517622	May 1996	Ivanoff et al.	395/200.13

ART-UNIT: 232

PRIMARY-EXAMINER: Harrity; John E.

ATTY-AGENT-FIRM: Elman & Associates

ABSTRACT:

The present invention provides a virtual network, sitting "above" the physical connectivity and thereby providing the administrative controls necessary to link various communication devices via an Access-Method-Independent Exchange. In this sense, the Access-Method-Independent Exchange can be viewed as providing the logical connectivity required. In accordance with the present invention, connectivity is provided by a series of communication primitives designed to work with each of the specific communication devices in use. As new communication devices are developed, primitives can be added to the Access-Method-Independent Exchange to support these new devices without changing the application source code. A Thread Communication Service is provided, along with a Binding Service to link Communication Points. A Thread Directory Service is available, as well as a Broker Service and a Thread Communication Switching Service. Intraprocess, as well as Interprocess, services are available. Dynamic Configuration Management and a Configurable Application Program Service provide software which can be commoditized, as well as upgraded while in operation.

2 Claims, 154 Drawing figures

Previous Doc Next Doc Go to Doc#

L20: Entry 6 of 12 File: USPT Dec 15, 1998

DOCUMENT-IDENTIFIER: US 5850518 A
TITLE: Access-method-independent exchange

Abstract Text (1):
The present invention provides a virtual network, sitting "above" the physical connectivity and thereby providing the administrative controls necessary to link various communication devices via an Access-Method-Independent Exchange. In this sense, the Access-Method-Independent Exchange can be viewed as providing the logical connectivity required. In accordance with the present invention, connectivity is provided by a series of communication primitives designed to work with each of the specific communication devices in use. As new communication devices are developed, primitives can be added to the Access-Method-Independent Exchange to support these new devices without changing the application source code. A Thread Communication Service is provided, along with a Binding Service to link Communication Points. A Thread Directory Service is available, as well as a Broker Service and a Thread Communication Switching Service. Intraprocess, as well as Interprocess, services are available. Dynamic Configuration Management and a Configurable Application Program Service provide software which can be commoditized, as well as upgraded while in operation.

Brief Summary Text (9):
As shown in FIG. 10, the lowest layer defined by the OSI model is called the "physical layer," and is concerned with transmitting raw data bits over the communication channel. Design of the physical layer involves issues of electrical, mechanical or optical engineering, depending on the medium used for the communication channel. The second layer, next above the physical layer, is called the "data link" layer. The main task of the data link layer is to transform the physical layer, which interfaces directly with the channel medium, into a communication link that appears error-free to the next layer above, known as the network layer. The data link layer performs such functions as structuring data into packets or frames, and attaching control information to the packets or frames, such as checksums for error detection, and packet numbers.

Brief Summary Text (11):
The basic function of the Transmission Control Protocol (TCP) is to make sure that commands and messages from an application protocol, such as computer mail, are sent to their desired destinations. TCP keeps track of what is sent, and retransmits anything that does not get to its destination correctly. If any message is too long to be sent as one "datagram," TCP will split it into multiple datagrams and makes sure that they all arrive correctly and are reassembled for the application program at the receiving end. Since these functions are needed for many applications, they are collected into a separate protocol (TCP) rather than being part of each application. TCP is implemented in the "transport layer," namely the fourth layer of the OSI reference model.

Brief Summary Text (13):
As shown in FIG. 10, the OSI model provides for three layers above the transport layer, namely a "session layer," a "presentation layer," and an "application layer," but in the Internet these theoretical "layers" are undifferentiated and generally are all handled by application software. The present invention provides

for session control and for communicating with applications programs. Thus the present invention may be described in accordance with the OSI theoretical model as operating at the session layer and application layers.

Brief Summary Text (17):
The present invention provides a virtual network, sitting "above" the physical connectivity and thereby providing the administrative controls necessary to link various communication devices via an Access-Method-Independent Exchange. In this sense, the Access-Method-Independent Exchange can be viewed as providing the logical connectivity required. In accordance with the present invention, connectivity is provided by a series of communication primitives designed to work with each of the specific communication devices in use. As new communication devices are developed, primitives can be added to the Access-Method-Independent Exchange to support these new devices without changing the application source code. When viewed in accordance with the OSI model, the communication primitives operate at the level of the transport layer, and, to the extent appropriate, at the network layer, and in some instances down to the data link layer, and occasionally as needed, the physical layer.

Brief Summary Text (20):
When users want to access the Access-Method-Independent Exchange, they simply supply Exchange with their unique identifiers. The Binding Service validates each user and permits access to the Exchange. The user may then connect to any registered service by simply calling the service's communication identifier. Of course, if they are unfamiliar with the service providers communication identifier, they can request assistance through the Thread Directory Service. The Thread Directory Service provides a listing of available services grouped by relationship. The user can search for keywords, titles, or other information such as service fees. Ultimately, the user can request to gain access to the service.

Brief Summary Text (21):
The Access-Method-Independent Exchange is not limited to servicing a particular geographic area and hence can easily work with foreign countries. The Access-Method-Independent Exchange includes the ability to provide voice or data message processing.

Brief Summary Text (23):
At the core of the technology is the Thread Communication Service (TCS), a software utility used to administer the dynamic communications between computer processes executing on a local computer, or, on a remote system. Two versions of the TCS have been implemented: one for intraprocess communications and a second for interprocess communications. The intraprocess version of the TCS is used for a single application process with multiple threads of control. The interprocess version of the TCS provides the administration of communications between processes executing in disjoint address spaces on a local, or remote computer system.

Brief Summary Text (36):
Communication primitives are registered with the Thread Communication Service for the specific operating system the TCS is executing on. The name, the location, and certain characteristics describing the communication primitive are retained by the TCS for subsequent use. In this context, the communication primitives become a reusable asset, needing to be developed and tested only one time.

Brief Summary Text (53):
When a process is executing, it may request the TCS to connect it to a communication point. For the intraprocess communication version of the TCS, the service executes as a separate thread of control. In the interprocess communication version of the TCS, the service executes as a separate process.

Brief Summary Text (58):

On systems supporting multiple threads of control within a single process address space, the TCS uses a special communication point called the intra.sub.-- p communication point to execute commands on behalf of the TCS within that address space. That is to say, when the application process makes its initial request to the TCS, the intra.sub.-- p communication point will bootstrap itself as a communication point within the application process address space. The TCS then issues specific commands to the intra.sub.-- p communication point who executes these commands on behalf of the TCS. The use of the intra.sub.-- p communication point is essential to permit intraprocess communication points while supporting interprocess communication points at the same time.

Brief Summary Text (59):
When an application makes a request to connect with a registered communication point, and that point must execute as a separate thread of control within the address space of the requesting process, then the TCS must have a method to satisfy the request. Since the TCS itself is executing in a different address space, it needs a worker thread executing within the requesting process's address space to spawn the requested communication point thread on its behalf.

Brief Summary Text (69):
When a communication point is registered, the communication point may have a specific communication primitive required to either send or receive message. This poses a challenge for another communication point to connect if the requesting communication point requires a different communication primitive. When this happens, the TCS will search for a broker communication point which can convert the messages from the first primitive type to the second primitive type. The broker service, if necessary, will be inserted between the requesting communication point and the requested service communication point.

Brief Summary Text (80):
7. Keyword search list used to locate this service entry

Brief Summary Text (102):
A process can request information from the Thread Directory Service specifying the desired search criteria. This is similar to dialing 411 and requesting a telephone number for an individual based on their name and street address. Each TDS has its own unique identifier. The registered communication points are assigned unique communication identifiers based on the TDS's identifier. Thus, communication points are fixed in the universe in this sense.

Brief Summary Text (109):
The Application Process constructs the Dynamically Configured Application Program in the Dynamic Configuration Management (DCM) by specifying zero or more RULES identifying the components of the Dynamically Configured Application Program, the interactions between these components, the policy for evaluating these components, the order of evaluation of these components, and a method for satisfying the RULE. The Application Process can specify zero or more data files referred to as Virtual Program Rules Files containing RULES for the Dynamically Configured Application Program. In this sense, the Application Process provides the blueprint for constructing the Dynamically Configured Application Program.

Brief Summary Text (128):
After evaluating the last Flow Rule of the Dynamically Configured Application Program, the DCM considers the application as having completed and returns control back to the initial Application Process.

Drawing Description Text (21):
FIG. 13.F Header File Declaring Binding Method.

Drawing Description Text (23):

FIG. 13.G Examples of Pattern, Transformation, Locate, Status and Query for a Shared Object in a Shared Library.

Drawing Description Text (25):
FIG. 13.I Example of Shared Library Binding Method including Pattern, Transformation, Locate, Status & Query.

Drawing Description Text (26):
FIG. 13.J Example of Data Structures Header File.

Drawing Description Text (34):
FIG. 15.A Communication Data Header File.

Drawing Description Text (37):
FIG. 16.A Compoints Header File.

Drawing Description Text (40):
FIG. 17.A Communications Registration Header File.

Drawing Description Text (42):
FIG. 17.C Communications Point Header File.

Drawing Description Text (45):
FIG. 18.A Thread Condition Variable Header File.

Drawing Description Text (48):
FIG. 19.A Generic Compoint Header File.

Drawing Description Text (51):
FIG. 20.A Thread Link List Header File.

Drawing Description Text (54):
FIG. 21.A Mutex Thread Log Header File.

Drawing Description Text (57):
FIG. 22.A Thread Mutex Header File.

Drawing Description Text (60):
FIG. 23.A Communication Primitive Header File.

Drawing Description Text (62):
FIG. 23.C Communication Primitive Data Header File.

Drawing Description Text (65):
FIG. 24.A Thread Queue Condition Header File.

Drawing Description Text (68):
FIG. 25.A Registry Header File.

Drawing Description Text (73):
FIG. 27.A Thread Reader-Writer Lock Header File.

Detailed Description Text (11):
The Application Program is therefore a static representation of a well-known functionality and is not easily able to dynamically load additional Threads unknown at the time the Application Program was developed. There are, however, certain Applications Programs which provide a listing of installed computer Application Programs either through a textual display or through a graphical representation referred to as an icon. Additionally, certain Application Processes search specific directories for available Application Programs to execute as Application Co-

Processes, but again, the criteria for their representation is static and unalterable by the end user.

Detailed Description Text (14):
A limitation of both the textual and graphical representation of the available Application Programs is that the information displayed to the user is dependent on the underlying operating system implementation. Certain operating systems will display the name, the size in bytes, the owner, the date created, and execution mode while others will display a subset of this information and possibly other system-dependent information. Regardless, however, the user cannot easily associate additional information with the installed application in a useful manner. Finally, many users have manually created what has become known as README files to describe this information.

Detailed Description Text (27):
Here, the source code provides the necessary algorithm, logic, and instructions in a human-readable form. This source code must then be compiled into an Application Program which the computer can then load and execute. The process of determining the necessary actions to create the Application Program are typically controlled by a software construction Application Program (a "make" utility) which reads specifications from a data file known as a "makefile". This process is well known and well understood in the computer programming profession. The makefile contains specification of the form:

Detailed Description Text (40):
2. All of the Application Program's Minor Services are developed and compiled into the Application Program which is then sold to the customer. Certain Minor Services, however, will be inaccessible to the customer unless the customer pays additional fees. At that time, a key file is provided to the customer with the appropriate purchased Minor Services turned on. I shall refer to this as "Run-Time Featuring."

Detailed Description Text (43):
Application Programs are typically designed and distributed following the Non-featuring Software model. Consider, for example, that when purchasing a Word Processing Application you receive all of the latest features available. This has the disadvantage that you are paying for features which you may not need.

Detailed Description Text (52):
With shared libraries, an application program references services available in the library without copying all of the text portion into the Application Program. When the Application Program is executed, the resulting Application Process opens the shared library, loads the service from the library, and then executes the service. The service is retained until the Application Process explicitly request that the service is to be removed from the Application Process. The advantage of using shared libraries is that the underlying library can be upgraded, altered, or otherwise changed independently of the Application Program.

Detailed Description Text (53):
The disadvantage in using shared libraries in this manner is that the shared library can only be altered when there are no Application Processes referencing the shared library. Another disadvantage in using shared libraries is that Application Programs are not normally designed to explicitly search and load services from the shared libraries on demand.

Detailed Description Text (59):
3. query the directory to search for services.

Detailed Description Text (62):
Once registered, any thread can call the Thread Directory Service to query for a Thread Service by providing one or more Public Attributes. The Thread Directory

Service will then search the Thread Service Directory reporting the Thread Communication Identifier(s) of those services matching the specified attributes. In querying the Thread Service Directory, a requesting thread can specify the search criteria attributes using Boolean expressions.

Detailed Description Text (86):
The Binding Service provides a series of Default Binding Methods including the ordering of Binding Methods as should be applied by the Binding Service. These Binding Methods and their ordering are specified in a Binding Configuration File which is read by the Binding Service during its initialization. Additional Binding Methods can be added to the Binding Configuration File by the end user. Other Binding Methods can be registered with the Binding Service during the Application Process' run time execution. The registration of a Binding Method must include the information shown in Table 1.

Detailed Description Text (90):
Pattern Matching: if the arbitrary named representation matches the specified regular expression pattern, then apply the Locate Operation to determine if the named representation can be found. If the Pattern Matching Method is specified as NULL, then proceed as if the name was matched. If the arbitrary named representation does not match the specified regular expression pattern, then go to the next Binding Method.

Detailed Description Text (91):
Transformation: if the arbitrary named representation successfully completes the Pattern Matching operation, then apply the Transformation operation to the arbitrary named representation and use this transformed name for subsequent operations. If the Transformation operation is not specified for this Binding Method, then use the specified arbitrary named representation for subsequent operations.

Detailed Description Text (109):
Establish request the Binding Service to read a specified Binding Service Configuration File

Detailed Description Text (115):
The Application Process constructs a Dynamically Configured Application Program in the DCM by specifying a series of RULES identifying the components of the Dynamically Configured Application Program, the interactions between these components, the policy for evaluating these components, the order of evaluation of these components, and a method for satisfying the RULE. Additionally, the Application Process can specify zero or more data files referred to as Program Rules Files containing RULES for the Dynamically Configured Application Program. In this sense, the Application Process provides the blueprint for constructing the Dynamically Configured Application Program either through an Application Programming Interface and through zero or mode Application Program Rules Files. Once constructed, the Application Process can then request the DCM to execute the Dynamically Configured Application Program.

Detailed Description Text (124):
After evaluating the last Flow Rule of the Dynamically Configured Application Program, the DCM considers the application as having completed and returns control back to the initial Application Process.

Detailed Description Text (131):
FUNCTION BINDING METHOD: The FUNCTION BINDING METHOD associates the rule name with a function (procedure) available in the application program. The DCM will search the symbol name list for the Application Process to locate the address of the function. If the DCM must trigger the method for the rule, then the function is invoked.

Detailed Description Text (132):
FILE BINDING METHOD: The rule name represents the name of a file accessible by the computer system.

Detailed Description Text (133):
DEFAULT BINDING METHOD: If no binding method has not been specified for the rule, then the DCM will bind the rule name using the DEFAULT BINDING METHOD. The DEFAULT BINDING METHOD is to associate the rule name with a function (procedure) available in the application program. The DCM will search the symbol name list for the Application Process to locate the address of the function. If the DCM must trigger the method for the rule, then the function is invoked. If the DCM cannot locate the function in the Application Process's symbol table, then the RULE is considered to have failed.

Detailed Description Text (190):
7. keyword search list used to locate this service entry;

Detailed Description Text (214):
A thread can query the information in the Thread Service Directory by calling the Thread Directory Service specifying a QUERY operation and providing zero or more components of an entry on which the Thread Service Directory is to search for. This information is then made available to the requesting thread.

Detailed Description Text (256):
The list of Communication Primitives available to the Application Process can be retained in memory, or, retained in a file on a storage medium (disk), or, retained in the TDS, or, retained in an Application Process accessible to the requesting Application Process.

Detailed Description Text (270):
The list of registered communication points can be retained in memory, or, retained in a file on a storage medium accessible to the computer system, or, retained in the TDS, or, retained in an Application Process accessible to the requesting Application Process.

Detailed Description Text (276):
2) the Application Process calls the TDS to search for a communication point based on specific criteria, or,

Detailed Description Text (281):
If necessary (based on registration data; see the section on TDS), the TCS will start an invocation of a communication point as a separate thread of control. The invocation of a communication point can be as a separate process, or, as a separate thread. In either case, the TCS will begin the invocation of the Minor Service if the Minor Service must be executed, and if there are no administrative constraints, such as the number of instances of runnable Minor Services of the specified type has been exhausted (See the section on TDS).

Detailed Description Text (299):
4) using the located communication point information, start an instance of the communication point and notify it of the communication control structure as defined above.

Detailed Description Text (339):
FIG. 15.A contains the communication data header file used by the communication data module with the example TCS Application Program.

Detailed Description Text (341):
FIG. 16.A contains the communication point header file used by the communication

point module with the example TCS Application Progra,

Detailed Description Text (343):
FIG. 17.A contains an example communication registration header file used by the communication registration module with the example TCS Application Program.

Detailed Description Text (345):
FIG. 17.C contains the example compoint header file used by the example compoint module with the example TCS Application Program.

Detailed Description Text (347):
FIG. 18.A contains an example thread condition variable header file used by the example thread condition variable module with the example TCS Application Program.

Detailed Description Text (349):
FIG. 19.A contains an example generic communication point header file for use with the example TCS Application Program.

Detailed Description Text (351):
FIG. 20.A contains an example thread link list header file for use with the example TCS Application Program.

Detailed Description Text (353):
FIG. 21.A contains an example of a mutex thread log header file for use with the example TCS Application Program.

Detailed Description Text (355):
FIG. 22.A contains an example of a thread mutex module header file for use with the thread mutex module with the example TCS Application Program.

Detailed Description Text (357):
FIG. 23.A contains an example of a communication primitive header file for use with the communication primitive module listed in FIG. 23.B.

Detailed Description Text (359):
FIG. 23.C contains an example of a communication primitive's data header file for use with the communication primitive's data module described in FIG. 23.D for use with the example TCS Application Program.

Detailed Description Text (361):
FIG. 24.A contains an example of a thread queue condition header file for use with the thread queue condition module described in FIG. 24.B for use with said example TCS Application Program.

Detailed Description Text (363):
FIG. 25.A contains an example of a registry header file for use with the registry module described in FIG. 25.B for use with the said example TCS Application Program.

Detailed Description Text (366):
FIG. 27.A provides an example of a thread reader-writer lock header file for use with the thread reader-writer module described in FIG. 27.B for use with the said example TCS Application Program.

Detailed Description Text (369):
The Thread Service Directory Communication Link provides the ability for a Thread Communication Link between Application Threads and Service Threads. A Service Thread may establish additional Thread Communication Links with other Service Threads, or Application Threads. In this context, each thread is said to be a Thread Communication Point (TCP). Note: As used in this section, the term TCP does

NOT refer to "Transmission Control Protocol." In its simplest example, this is shown in FIG. 1 in which the Thread Communication Point labeled TCP-1 is connected to the Thread Communication Point labeled TCP-2 through a Thread Communication Link (TCL). See FIG. 1.

Detailed Description Text (382):
The Binding Service provides a series of Default Binding Methods including the ordering of Binding Methods as should be applied by the Binding Service. These Binding Methods and their ordering are specified in a Binding Configuration File which is read by the Binding Service during its initialization. Additional Binding Methods can be added to the Binding Configuration File by the end user. Other Binding Methods can be registered with the Binding Service during the Application Process's run time execution. The registration of a Binding Method must include the information shown in Table 6.

Detailed Description Text (388):
Pattern Matching: if the arbitrary named representation matches the specified regular expression pattern, then apply the Locate Operation to determine if the named representation can be found. If the Pattern Matching Method is specified as NULL, then proceed as if the name was matched. If the arbitrary named representation does not match the specified regular expression pattern, then go to the next Binding Method.

Detailed Description Text (389):
Transformation: if the arbitrary named representation successfully completes the Pattern Matching operation, then apply the Transformation operation to the arbitrary named representation and use this transformed name for subsequent operations. If the Transformation operation is not specified for this Binding Method, then use the specified arbitrary named representation for subsequent operations.

Detailed Description Text (407):
Establish: request the Binding Service to read a specified Binding Service Configuration File

Detailed Description Text (413):
The Binding Service, upon initialization will read a Binding Service Configuration File containing zero or more entries of the form:

Detailed Description Text (419):
The Application Process uses the Binding Service's Register interface to register additional Binding Methods not specified in a Configuration File. The specification for the new Binding Method requires the same data as that provided through the Configuration Files. The Register interface provides a mechanism to specify additional Binding Methods even after the Binding Service has started execution.

Detailed Description Text (426):
The first step in applying the Bind interface is to search the table to see if the specified arbitrary named representation has already been BOUND. If not, then the Binding Service will create a new entry in the table for the specified arbitrary named representation. This entry includes the arbitrary named representation and initially is marked as unbound. If the specified arbitrary named representation is already listed in the table and is marked as BOUND, then the Bind interface completes successfully without further action being required. If the specified arbitrary named representation is already listed in the table, but is marked as UNBOUND, then the Bind interface continues.

Detailed Description Text (429):
During the evaluation of the Binding Methods, the Binding Service will first examine the Binding Method to see if it has an associated Pattern Matching

operation. If so, then the arbitrary named representation is compared against the specified expression pattern using the specified Pattern Matching operation. If the arbitrary name does not compare with the specified regular expression pattern then the next Binding Method is applied. If the Pattern Matching operation is specified as NULL, or if the expression pattern is specified as NULL, or if the arbitrary named representation compares with the specified pattern, then the arbitrary named representation is considered for binding to this Binding Method.

Detailed Description Text (430):
Upon successfully completing the Pattern Matching operation, the Binding Service applies the Name Transformation operation, if specified for the Binding Method. The Name Transformation operation is used to generate an alternative named representation which will then be used for the Locate and the Status operations described below.

Detailed Description Text (431):
When the arbitrary named representation is considered for binding to a Binding Method, then the Binding Service will invoke the Binding Method's Locate operation to locate the entity associated with the alternative named representation, or the arbitrary named representation if the alternative is not defined. If the Locate operation fails and a pattern was specified, then the arbitrary named representation is considered BOUND, but not found. Otherwise, the arbitrary named representation remains UNBOUND.

Detailed Description Text (443):
The Application Process constructs a Dynamically Configured Application Program in the DCM by specifying a series of RULES identifying the components of the Dynamically Configured Application Program, the interactions between these components, the policy for evaluating these components, the order of evaluation of these components, and a method for satisfying the RULE. Additionally, the Application Process can specify zero or more data files referred to as Application Program Rules Files containing RULES for the Dynamically Configured Application Program. In this sense, the Application Process provides the blueprint for constructing the Dynamically Configured Application Program either through an Application Programming Interface and through zero or mode Application Program Rules Files. Once constructed, the Application Process can then request the DCM to execute the Dynamically Configured Application Program.

Detailed Description Text (455):
After evaluating the last Flow Rule of the Dynamically Configured Application Program, the DCM considers the application as having completed and returns control back to the requesting Application Process.

Detailed Description Text (461):
Initially when a RULE is specified, the DCM makes no assumptions as to what the RULE name represents. During the evaluation of the RULE, the DCM uses the Binding Service to associate the RULE name with an entity understood by the DCM. The list of entities understood by the DCM and their corresponding interpretation by the DCM are provided during the initialization of the DCM. The list of entities appear in the Binding Service's Configuration File. In this sense, the list of entities can be modified and updated over time based on market demand for new entities and their interpretations. Initially, the DCM provides the Binding Methods for:

Detailed Description Text (466):
FUNCTION BINDING METHOD: The FUNCTION BINDING METHOD associates the rule name with a function (procedure) available in the application program. The DCM will search the symbol name list for the Application Process to locate the address of the function. If the DCM must trigger the method for the rule, then the function is invoked.

Detailed Description Text (467):
FILE BINDING METHOD: The rule name represents the name of a file accessible by the computer system.

Detailed Description Text (468):
DEFAULT BINDING METHOD: If no binding method has not been specified for the rule, then the DCM will bind the rule name using the DEFAULT BINDING METHOD. The DEFAULT BINDING METHOD is to associate the rule name with a function (procedure) available in the application program. The DCM will search the symbol name list for the Application Process to locate the address of the function. If the DCM must trigger the method for the rule, then the function is invoked. If the DCM cannot locate the function in the Application Process's symbol table, then the RULE is considered to have failed.

Detailed Description Text (472):
In executing the report rule, the DCM will first evaluate the prerequisite "spellCheck" rule. In evaluating the spellCheck rule, the DCM will first evaluate the libspell.so rule. In evaluating the libspell.so rule, the DCM will not find any prerequisites of the rule and will therefore attempt to bind the libspell.so rule name. The binding method will match using the SHARED LIBRARY BINDING METHOD and the name libspell.so will be BOUND to a shared library. In evaluating the SHARED.OBJECT rule, the DCM will recognize that the spellCheck rule name is to be interpreted as a function available in the shared library libspell.so. If the method for the rule is to be executed, then the spellCheck shared object is extracted from the shared library and is invoked as a function of the Dynamically Configured Application Program.

Detailed Description Text (476):
A Word Processing Application Program is designed. Using a standard software construction utility, converting the source code to the Application Program which in turn is sold to end-users. After the initial market acceptance of the Word Processing Application Program, additional features are developed including a Thesaurus, a Spell Checking Feature and a Grammar Checking Feature. Each of these features is designed, developed, implemented and sold as separate Minor Services for the Word Processing Application Service.

Detailed Description Text (477):
When a user purchases the Word Processing Application Program, the installation process creates a Application Program Rules file for the Word Processing Application Service. The Rules file contains:

Detailed Description Text (487):
The Application Program Rules file is then given to the DCM to establish the rules for the Dynamically Configured Application Program. When executed, the APINIT rule is evaluated, which causes the SetEnvironment rule to be evaluated which in turn causes the setEnv rule to be evaluated. The setEnv rule has a THREAD prerequisite causing the DCM to begin the procedure named setEnv as a thread of execution within the Application Process. Similarly, the wp and clearEnv rules, when evaluated, will cause the DCM to create a thread of execution within the Application Process for their named procedures.

Detailed Description Text (488):
The user subsequently purchases the Spell Checker feature and the Thesaurus, but not the Grammar Checking Feature. The installation of the Spell Checking Feature and the Thesaurus Feature cause the Application Program Rules file for the Word Processing Application Service to be modified as follows:

Detailed Description Text (526):
The LOCATE operation instructs the Configurable Application Process Service to locate a specified AMS. The default action performed by the CAMS is to search

through the current Application Process for the specified AMS, and if not found, to search through zero or more specified shared libraries. Once located, the CAMS returns the location of the specified AMS.

Detailed Description Text (555):
The REPLACE operation specifies that the Configurable Application Process Service is to replace a specified ACTIVE AMS with a different specified AMS. The existing AMS must have been started using the EXECUTE operation (thus must appear on the ACTIVE AMS List). The REPLACE operation searches the ACTIVE AMS List for the specified AMS to replace. If not found, then the operation is aborted. The new AMS replacing the existing AMS must have previously been configured with the LOAD operation and must therefore appear in the LOADED AMS List.

Detailed Description Text (585):
The NOTIFICATION operation specifies to the Configurable Application Process Service that a particular internal or external event has occurred. The Configurable Application Process Service searches the EVENT AMS List to determine the any actions it is to perform based on the receipt of this event, and then performs these actions.

Detailed Description Text (617):
The Named Execution Environment Configuration File

Detailed Description Text (619):
Multiple REGISTERED ATTRIBUTES SETS can be registered for a single computer system. The REGISTERED ATTRIBUTES, along with the name of the computer system are stored in memory, or on a storage media accessible to the computer system on which the NEEM is executing. A permanent copy of the REGISTERED ATTRIBUTES along with their computer system name can be stored in the Named Execution Environment Configuration File, hereinafter referred to as the NEECF. stored in memory, or on a storage media accessible to the computer system on which the NEEM is executing. A permanent copy of the REGISTERED ATTRIBUTES along with their computer system name can be stored in the Named Execution Environment Configuration File, hereinafter referred to as the NEECF.

Detailed Description Text (641):
The NEEM searches the REGISTERED ATTRIBUTE entries for an environment satisfying the REQUESTED ATTRIBUTES set forth by the Application Process. When a match has been found, then the NEEM will establish a shell on the associated computer system to execute the requests of the Application Process. The newly created environment then becomes an ACTIVE NEE, and is added to the ACTIVE NEE List maintained by the NEEM.

Detailed Description Text (643):
Just prior to executing the shell Application Process associated with a new ACTIVE NEE, the NEEM will arrange for the standard input, the standard output, and the standard error file descriptors to be redirected to/from the NEEM itself. In this manner, the NEEM will be able to appropriately redirect all I/O requests on this file descriptors appropriately. The NEEM will also establish Minor Service Reader/Writer threads to read the standard output and standard error file descriptors and to redirect this output to the appropriate Application Process making a request to execute in the ACTIVE NEE. This is shown in FIG. 8.

Detailed Description Text (647):
There are a series of commands understood by the NEEM through which the Application Process can control the use of a logically named NEE. These include:

Detailed Description Paragraph Table (3):
TABLE 2 _____ Default Binding Methods.
_____ File Binding Method a method to bind the

arbitrary named representation to a file accessible by the computer system Shell Binding Method a method to bind the arbitrary named representation to a user level shell Data Binding Method a method to bind the arbitrary named representation to a datum available to the Application Process Function Binding Method a method to bind the arbitrary named representation to a function (procedure) accessible to the Application Process Thread Binding Method a method to bind the arbitrary named representation to a thread of the Application Process Process Binding Method a method to bind the arbitrary named representation to an existing Application Process

Detailed Description Paragraph Table (4):
TABLE 3 Binding Method Operations
Pattern Matching Method Name Transformation Method Locate Method Status Method Query Method

Detailed Description Paragraph Table (8):
TABLE 7 Default Binding Methods.
File Binding Method a method to bind the arbitrary named representation to a file accessible by the computer system Shell Binding Method a method to bind the arbitrary named representation to a user level shell Data Binding Method a method to bind the arbitrary named representation to a datum available to the Application Process Function Binding Method a method to bind the arbitrary named representation to a function (procedure) accessible to the Application Process Thread Binding Method a method to bind the arbitrary named representation to a thread of the Application Process Process Binding Method a method to bind the arbitrary named representation to an existing Application Process

Detailed Description Paragraph Table (9):
TABLE 8 Binding Method Operations.
Pattern Matching Method Name Transformation Method Locate Method Status Method Query Method

Previous Doc Next Doc Go to Doc#